

SDHCAL simulation status and how to run it

Guillaume Garillot

- The SDHCAL prototype simulation consists in 2 tools :
 - A Geant4 simulation that simulates propagation and interaction of particles in the SDHCAL
 - A MarlinReco package (SimDigital) that modelises RPC response

- To install :
 - git clone <https://github.com/SDHCAL/SDHCALSim>
 - cd SDHCALSim
 - mkdir build ; cd build
 - cmake -C \$ILCSOFT/ILCSoft.cmake
 - make install
- Run a simple example :
 - modify line 12 the script/example.py and put the correct path
 - launch example.py
 - it will simulate 10 10GeV electrons

- To Create your own script
- Global parameters :
 - physicsList , nEvent , seed ,
outputFileName

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'
```

- To Create your own script
- Add particle gun :
 - set particleName

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
```

- To Create your own script
- Add particle gun :
 - particle gun position options :
 - fixed :
 - fixed position defined by :
 - positionX , positionY , positionZ (in mm)

```
#!/usr/bin/env python

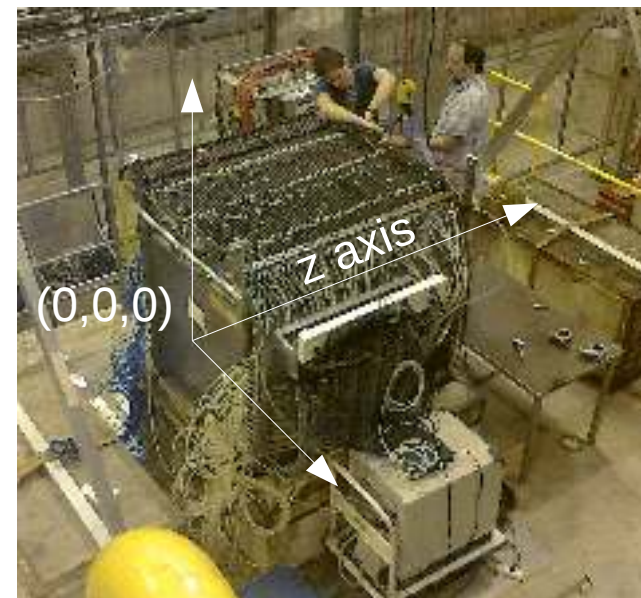
import os
import sys
import SDHCALSim as sim

if __name__ == '__main__':

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'fixed'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
```



- To Create your own script
- Add particle gun :
 - particle gun position options :
 - uniform :
 - 2D square in the XY plane centered on positionX , positionY with half width uniformDeltaPos

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'uniform'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.uniformDeltaPos = 10
```

- To Create your own script
- Add particle gun :
 - particle gun position options :
 - gaus :
 - 2D gaussian in the XY plane centered on positionX , positionY with sd sigmaPos

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10
```


- To Create your own script
- Add particle gun :
 - particle gun position options :
 - cosmic :
 - launch particles from everywhere with all possible angles
 - mainly used with muons for multiplicity study

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'mu-'
    particle.positionOption = 'cosmic'
```

- To Create your own script
- Add particle gun :
 - particle gun momentum options :
 - (momentum means only direction here, not the relativistic quantity)
 - fixed :
 - launch all the particles along the axis defined by momentumPhi and momentumTheta
 - $\theta = 0$ is aligned with z axis

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10

    particle.momentumOption = 'fixed'
    particle.momentumPhi = 0
    particle.momentumTheta = 0
```

- To Create your own script
- Add particle gun :
 - particle gun momentum options :
 - (momentum means only direction here, not the relativistic quantity)
 - gaus :
 - launch all the particles in a conic distribution centered along the axis defined by momentumPhi and momentumTheta
 - the openness of this conic distribution can be set using the sigmaMomentum parameter

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10

    particle.momentumOption = 'gaus'
    particle.momentumPhi = 0
    particle.momentumTheta = 0
    particle.sigmaMomentum = 5
```

- To Create your own script
- Add particle gun :
 - particle gun momentum options :
 - (momentum means only direction here, not the relativistic quantity)
 - if you used cosmic position option you don't have to care about momentum settings

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'mu-'
    particle.positionOption = 'cosmic'
```

- To Create your own script
- Add particle gun :
 - particle gun energy options :
 - fixed :
 - all the particles from this gun will have the same energy defined by the energy parameter (in GeV)

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10

    particle.momentumOption = 'gaus'
    particle.momentumPhi = 0
    particle.momentumTheta = 0
    particle.sigmaMomentum = 5

    particle.energyDistribution = 'fixed'
    particle.energy = 50
```

- To Create your own script
- Add particle gun :
 - particle gun energy options :
 - uniform :
 - energy of particles uniformly distributed between minEnergy and maxEnergy
 - convenient for MVA training

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10

    particle.momentumOption = 'gaus'
    particle.momentumPhi = 0
    particle.momentumTheta = 0
    particle.sigmaMomentum = 5

    particle.energyDistribution = 'uniform'
    particle.minEnergy = 1
    particle.maxEnergy = 120
```

- To Create your own script
- Add the particle gun to the global params
- Then launch the sim using this params
- You can now execute your script

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle = sim.Particle()
    particle.particleName = 'pi-'
    particle.positionOption = 'gaus'
    particle.positionX = 0
    particle.positionY = 0
    particle.positionZ = -20
    particle.sigmaPos = 10

    particle.momentumOption = 'gaus'
    particle.momentumPhi = 0
    particle.momentumTheta = 0
    particle.sigmaMomentum = 5

    particle.energyDistribution = 'fixed'
    particle.energy = 50

    params.particleList.append( particle )

    sim.launch( params )
```

- Example if you want to launch two particles for each event :
- You can delay a particle using the time parameter

```
#!/usr/bin/env python

import os
import sys
import SDHCALSim as sim

if __name__ == '__main__' :

    os.environ["SIMEXE"] = '/path/to/SDHCALSim/bin/SDHCALSim'

    params = sim.Params()
    params.physicsList = 'QGSP_BERT'
    params.nEvent = 10
    params.seed = 0
    params.outputFileName = 'outputFile'

    particle1 = sim.Particle()
    particle1.particleName = 'pi-'
    particle1.positionX = -10
    particle1.positionY = -10
    particle1.energy = 50

    particle1.time = 0

    particle2 = sim.Particle()
    particle2.particleName = 'neutron'
    particle2.positionX = 10
    particle2.positionY = 10
    particle2.energy = 20

    particle2.time = 5

    params.particleList.append( particle1 )
    params.particleList.append( particle2 )

    sim.launch( params )
```


- Simulation output :

```

Event 0
Primary particles :

  pdgID : e-
  Time : 0 ns
  Pos : (-12.7654,22.6652,-20)
  Mom : (-0.80918461,0.80170711,0.999956)
  Energy : 10 GeV

Deposited energy : 9.98827 GeV
Leaked energy : 0.808510999 GeV
EM Fraction : 99.9933 %

computing time : 0.18722
average time : 0.18722
-----
Event 1
Primary particles :

  pdgID : e-
  Time : 0 ns
  Pos : (-23.9209,47.8567,-20)
  Mom : (0.8202289,-0.112249,0.993474)
  Energy : 10 GeV

Deposited energy : 9.95301 GeV
Leaked energy : 0.80353596 GeV
EM Fraction : 99.8286 %

computing time : 0.1742
average time : 0.18071
-----
Event 2
Primary particles :

  pdgID : e-
  Time : 0 ns
  Pos : (30.1543,-57.5725,-20)
  Mom : (-0.80461676,-0.0290372,0.999789)
  Energy : 10 GeV

Deposited energy : 9.99438 GeV
Leaked energy : 0.80561786 GeV
EM Fraction : 100 %

computing time : 0.168006
average time : 0.176475
-----

```

- The simulation produces 2 files :
 - A root file which contains debug variables
 - The slcio file which contains the SimCalorimeterHits
 - This slcio file has to be 'digitized' using SimDigital

- SimDigital : MarlinReco processor to simulate RPC response
- Included in ilcsoft since 01-19-05 version
- Steering script example :
 - /gridgroup/ilc/garillot/MarlinReco/script/steer.xml
- launch with :
 - Marlin steer.xml
- Produce a lcio file that has the same format as SDHCAL data
- It also contains a LCRelation collection to link hits with primary particles, to help identify hits from 2 nearby showers

- This example steering file sets up a completely uniform detector (constant efficiencies and multiplicities for all the RPCs)
- To simulate the multiplicities variations you need to apply this changes :

```
...
<!--Induced charge simulation parameters-->
<parameter name="PolyaOption" type="string"> PerAsic </parameter>
<parameter name="PolyaMapFile" type="string"> /gridgroup/ilc/garillot/files/DigitMap/Polya_730677.root </parameter>

...
...

<!--Induced charge dispatching : dispatching mode-->
<!-- Define the charge splitter method. Possible option : Erf , Exact , ExactPerAsic -->
<parameter name="ChargeSplitterOption" type="string"> PerAsic </parameter>
<parameter name="SpreaderMapFile" type="string"> /gridgroup/ilc/garillot/files/DigitMap/Polya_730677.root </parameter>

<!--Step efficiency correction method : should be Uniform or PerAsic-->
<parameter name="EffMapOption" type="string"> PerAsic </parameter>
<!--Value of the constant term for efficiency correction if EffMapOption==Uniform-->
<parameter name="EffMapConstantValue" type="float"> 0.96 </parameter>
<!--File name where prototype efficiency correction is stored if EffMapOption==PerAsic-->
<parameter name="EffMapFile" type="string"> /gridgroup/ilc/garillot/files/DigitMap/Eff_730677.root </parameter>
```

- The non uniform multiplicity is tuned to reproduce the October 2015 Test Beam, thus the agreement is not optimal on other test beams

- Other tools :
 - CaloSoftWare / SDHCALMarlinProcessor (originally coded by Arnaud Steen)
 - <https://github.com/ggarillot/CaloSoftWare>
 - <https://github.com/ggarillot/SDHCALMarlinProcessor>
 - (both are needed)